# Leveraging Knowledge Reuse and System Agility in the Outsourcing Era

**Igor Crk, Dane Sorensen, Amit Mitra, Amar Gupta**

## Abstract

Collaborative work groups that span multiple locations and time zones, or "follow the sun," create a growing demand for creating new technologies and methodologies that enable traditional spatial and temporal separations to be surmounted in an effective and productive manner. The hurdles faced by members of such virtual teams are in three key areas: differences in concepts and terminologies used by the different teams, differences in understanding the problem domain under consideration, and differences in training, knowledge, and skills that exist across the teams. These reasons provide some of the basis for the delineation of new architectural approaches that can normalize knowledge and provide reusable artifacts in a knowledge repository.

**KEYWORDS:** Knowledge Reuse, Agility, Offshore Outsourcing, Outsourcing, 24-Hour Knowledge Factory, Object Management, Metamodeling, Modeling, Software Factory

## 1 Introduction

The increasing prevalence of collaborative work groups that span multiple locations and time zones create a growing demand for creating new technologies and methodologies that can enable traditional spatial and temporal separations to be surmounted in an effective and productive manner. In the specific case of information technology (IT), more than 380,000 professionals are currently focused exclusively on export oriented activities (Aggarwal and Pandey, 2004). The hurdles faced by members of such virtual teams are in three key areas: (i) Differences in concepts and terminologies used by the different teams; (ii) Differences in understanding the problem domain under consideration; and (iii) Differences in training, knowledge, and skills that exist across the teams (P. Wongthongtham, E. Chang, T.S. Dillon, I. Sommerville 2006). These reasons

provide some of the basis for the delineation of new architectural approaches that can normalize knowledge and provide reusable artifacts in a knowledge repository.

This paper focuses on the issue of providing information systems agility, especially when the work is outsourced from one country (or company) to another or as the work is performed in multiple countries using a hybrid offshoring model such as the 24-Hour Knowledge Factory concept (Gupta, Seshasai, Mukherji, Ganguly 2007). This paper also deals with the issue of creating an evolving knowledge repository that can be used when systems need to be redesigned or re-implemented.

## 2 Related Work

The Object Management Group (OMG) is actively involved in the creation of a heterogeneous distributed object standard. In a departure from modeling standards, such as the Common Object Request Broker Architecture (CORBA) and the related Data Distribution Service (DDS), OMG moved towards the Unified Modeling Language (UML) and the related standards of Meta-Object Facility (MOF), XML Data Interchange (XMI), and Query Views Transformation (QVT). The latter standards provide a foundation for the Model Drive Architecture (MDA). In an effort to bring UML and the Semantic Web together, OMG is leading progress toward the Ontology Definition Metamodel.

More specifically, MDA, as related to software engineering, composes a set of guidelines for creating specifications structured as models. In MDA, functionality is defined using a platform-independent model with a domain-specific language. The domain specific language definition can be translated into platform-specific models by use of a Platform Definition Model (PDM). The Ontology Definition Metamodel is an OMG specification that links Common Logic and OWL/RDF ontologies with MDA. Common Logic being an ISO standard for facilitating the exchange of knowledge and information in computer-based systems, and Resource Description Framework (RDF) and Web Ontology Language (OWL) being the latest examples of framework and related markup languages

for describing resources authored by the World Wide Web Consortium (W3C). OMG and W3C standards are available online, at omg.org and w3.org, respectively.

Knowledge reuse has been previously explored with respect to organizational memory systems, or knowledge repositories. Markus (2001) identified distinct situations in which reuse arose according to the purpose of knowledge reuse and parties involved. The knowledge reuse situations exist among producers who reuse their own knowledge, those who share knowledge, novices seeking expert knowledge, and secondary knowledge miners. The solutions to the problems of meeting the requirements of knowledge storage or retrieval were presented as a combination of incentives and intermediaries.

In the context of allocation of IT resources, O'Leary (2001) conducted a case study of a knowledge management system of a professional service firm concluding that service-wise requirements for knowledge reuse should impact the design of knowledge systems. For example, the studied firm contained three primary service lines, tax, consulting, and audit. Differential reuse stemming from the relatively low reuse in the consulting service line to high reuse in the tax line, leads to a particular allocation of knowledge bases, software, hardware, and network resources.

O'Leary's paper supports earlier work by Vanwelkenhuysen and Mizoguchi (1995) which showed that knowledge reuse has at that point depended on organizational aspects of knowledge systems. Their work suggested dimensions along which ontologies for knowledge reuse may be built, based on workplace-adapted behaviors.

Knowledge reuse and agility is especially relevant to "follow the sun" models, similar in spirit to the 24-Hour Knowledge Factory, and have been attepted by others. Carmel (1999, pp. 27-32) describes one such project at IBM. In this early project, IBM established several offshore centers in a hub-and-spoke model where the Seattle office acted as the hub. Each offshored site was staffed by a phalanx, a mix of skill sets that were replicated across each spoke. Work would be handed out by the Seattle hub and each spoke would

accomplish the given task and send the results back to Seattle. This hub-and-spoke model necessitates specialization of the Seattle site. With only one site offering the specialized service, the Seattle site quickly became overwhelmed. The original goal of daily code drops could not be maintained.

A relevant case study involved the three distinct projects involving sites in the United States and India. These three projects were generally considered a failure. Treinen and Miller-Frost (2006) supply several lessons learned that are echoed in other studies, particularly problems with continuity, misunderstanding and the lag time between cycles of conversation. Cultural differences are also cited as problematic, especially with respect to various assumptions that were held in lieu of well specified requirements and planning.

Perhaps the most relevant study in respect to the 24-Hour Knowledge Factory, *Follow the Sun: Distributed Extreme Programming Development* (Yap 2005) describes a globally distributed, round-the-clock software development project. Here, a programming team was distributed across three sites (US, UK, and Asia) and they used collective ownership of code. One of the three sites already had knowledge of extreme programming. The two remaining sites were coached on extreme programming practices prior to the collaboration. These two sites believed that the first site had an advantage due to its previous knowledge with extreme programming. The three sites also met in person, which they felt helped the program start by building confidence in the members of other sites. The team used Virtual Network Computing (VNC) and video conferencing to facilitate communication. Hand-off of project artifacts initially consisted of a daily work summary, but later grew to include knowledge learned and new objectives.

Xiaohu, et. al. (2004) discusses the situation where development teams were dispersed globally, though, it seemed that each global unit was still responsible for its own module of the project. The teams did not need to discuss development decisions with each other unless they were related to interfacing or would affect another team. They used the

extreme programming method, but because of the global dispersal of teams, they lacked the benefits of customer co-location and participation. They thought the inability to get rapid customer feedback when the customer was in a different location adversely impacted the development of the product and the development time. These issues could easily impact development in a 24-Hour Knowledge Factory setting because customers in one location would thus not be able to interact with all sites.

The attempted, and particularly the failed, "follow the sun" approaches highlight the need for an agile knowledge ontology that more adequately manages the problem of change.

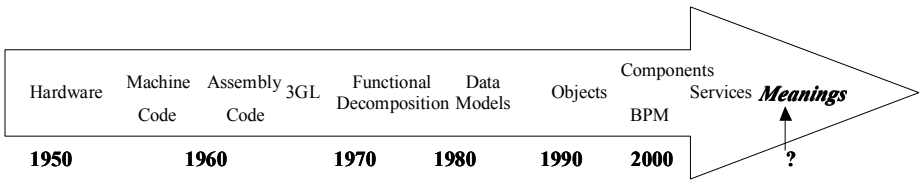
## 3 Agility and the Problem of Change

Change is difficult, complex and risky because it usually has unintended side effects. Each decision has many consequences, which in turn have many more. The Y2K problem is a classic example of a seemingly innocuous design decision that snowballed into a worldwide problem. The decision to use a two-digit representation of the year was originally deemed to be prudent. Later, it was thought to be a problem that would cripple computer systems when their clocks rolled over into the year 2000, since 00 is ambiguous. Ultimately, it cost the world around $600 billion (López-Bassols, Vladimir, 1998) to convert a two digit representation of the calendar year to four digits!

### 3.1 Fundamental Computing Technologies

Figure 1 shows the evolution of computing technology as researchers sought to tackle the problem of change and to remain agile though increasingly more complex demands are placed upon the technology.

Figure 1.




At the far left end of the spectrum lies hardware, originally physically and meticulously programmed to perform relatively simple tasks. Machine code replaced the physical

machine programming by the formulation of CPU-specific words, bit patterns corresponding to different commands that can be issued to the machine. Each type of CPU has its own machine code. Similarly, the CPU architecture has a corresponding assembly language. As such, assembly language is not portable and does not increase flexibility, but it does provide the essential abstractions that free the programmer from the tedium of remembering numeric codes or calculating addresses (as was the case when programming was accomplished through machine code). An assembly language is an example of a second-generation language. Third generation languages, denoted by 3GL in Figure 1, finally freed the task of programming from the underlying hardware. This is a much overlooked, but crucial, example of adapting technology to find a solution to the problem of change.

Since the time of the advent of 3GLs, programmers have been able to develop applications without understanding the underpinnings of the machines they worked on. Agility improved as programmers no longer needed to be hardware experts, their programming expertise remained even as the underlying hardware became obsolete. As we move along the timeline in Figure 1, we observe the adoption of new ideas that further built upon the flexibility of those preceding them. As abstraction increases, so does flexibility. The abstraction also allows for task modularity. With the advent of 3GLs, the underlying arcane did not disappear, but instead became encapsulated in the tools which make the 3GL abstraction possible, e.g. a C compiler or Java interpreter. Compilers and interpreters translate the high-level language constructs into machine or assembly code, thereby allowing for reuse of developed artifacts across any platform having a compatible compiler or interpreter.

The more recent notion of component-based development (CBD) involves building software systems using prepackaged software components (Ravichandran, T. 2005). CBD involves reusing application frameworks, which provide the architecture for assembling components into a software system. Components and frameworks may be either developed in-house or externally procured. CBD typically involves using both in-house developed and externally procured software components and frameworks. CBD

leverages the emergence of middleware and software objects standards to make software reuse a reality (Ravichandran, T. 2005). Since CBD encourages the move toward more modular systems built from reusable software artifacts, it is expected to enhance the adaptability, scalability, and maintainability of the resultant software (Szyperski, C 1997). CBD requires systems to be architected using a component framework necessitating developers to think through the interrelationships between various elements of an application system at a more granular level at an earlier stage in the development process than in traditional development approaches (Sparling, M 2000).

### 3.2 IBM's System/360: The Beginnings of Agile Development

A good example of a business transformation tackling the issues of agility and change through modularity is provided by IBM's System/360 (Amdahl and Blaauw, 2000) in the 1960's (Baldwin and Clark, 2000). The hardwired instruction sets of virtually all computers in the 1950s imposed a high level of interdependence of design parameters. Each computer was designed from scratch and each market niche was matched with a different system. Searching for new ways for teams to work together on a project, IBM led the effort to use modularity as a guiding principle. System/360 was the result of that effort. Further, System/360 marks the point at which the industry was transformed from a virtual monopoly to a modular cluster comprised of more than a thousand publicly traded firms and many startups (Fergusson 2004).

What makes System/360 an important landmark in the agility effort is that it belongs to the first family of computers that was designed with a clear distinction between architecture and implementation. The architecture of each model in the 360 family was introduced as an industry standard, while the system peripherals, such as disk drives, magnetic tape drives, or communication interfaces, allowed the customer to configure the system by selecting from this list. With the standardization of the architecture and peripheral interfaces, IBM opened the doors for the commodity component market. With its list of peripherals, System/360 allowed the technology to adapt to a customer's needs. Its backward compatibility tackled the problem of change in its own right, by allowing customers to upgrade and replace their hardware without losing essential capabilities. The

idea of encapsulation of functionality and the standardization of system interfaces inherent in System/360 is critical to understanding the importance of leveraging and reuse of knowledge.

## 4 Business Rules and the Structure of Information

Just as was the case with early computer technology decades ago, prior to 3GL in our first example and System/360 in the second, today's business rules are replicated in dissimilar formats in intermingled ways in multiple information systems and business processes. When any rule is changed, a concerted effort must be launched to make modifications in multiple systems. It makes change and innovation complex and error-prone. The framework described in this paper attempts to untangle business rules with an ontology derived from the inherent structure of information. By untangling business rules even in complex legacy models and systems, one gains the capability to represent specific elements of business knowledge once, and only once, in a knowledge repository. Using this repository, the specific elements of knowledge can be designed to naturally manifest themselves, in appropriate forms, to suit the idiosyncrasies of different business contexts.

As business processes became more tightly coupled with automation, the lack of agility in information systems became a serious bottleneck to product and process innovation. Frameworks that have attempted to solve this problem include: Structured Programming, Reusable Code Libraries, Relational Databases, Expert Systems, Object Technology, CASE tools, Code generators and CAPE tools. They were not very effective partially because they did not adequately address the ripple effects of change; ideally, business rules and knowledge should be represented so that when we change a rule once, corresponding changes should automatically ripple across all the relevant business processes (Gupta, A. Mitra, A. 2006).

Knowledge transfer and reuse (Myopolous 1998, Zyl and Corbett 2000, Kingston 2002) attain greater importance in the case of outsourcing. In order to achieve efficiency of resource consumption, we need new approaches to facilitate encapsulation of knowledge and the sharing of such knowledge among the relevant set of workers.

## 5 The Framework of Knowledge Reuse

Knowledge reuse is greatly aided by the existence of a knowledge repository; and the creation of a knowledge repository requires a means for the encapsulating information. Knowledge represents a coordinated set of information: rules of business, imposed by man or nature, either explicitly stated or implied. Knowledge can be described in terms of assertions and rules, allowing for the identification of constraints, issues, guidelines, and caveats.

While meaning and understanding are abstract notions, they are rooted in the physical world. We learned in chemistry that we can continually subdivide a substance before reaching a building-block, the subdivision of which would disallow us from identifying the substance and knowing its properties. Similarly, to identify the components of knowledge, we must distinguish between assertions whose division will involve no loss of information, and assertions whose division will sacrifice meaning: if an assertion is decomposed into smaller parts and the information lost cannot be recovered by reassembling the pieces. The fundamental rules that cannot be decomposed further without irrecoverable loss of information are called indivisible rules, atomic rules, or irreducible facts (Ross, R. G. (1997), Krifka, M. (WS 2000-2001)).

Irreducible facts embody pivotal information and constitute the root of coordinated requirements. These irreducible facts are woven together to create normalized knowledge. Normalized knowledge can then be utilized to coordinate complex activities in transnational corporations and hybrid offshore models such as the 24-Hour Knowledge Factory (24HrKF) discussed later in this paper. In the legacy systems of today, the process of making a single change opens up Pandora's Box, primarily because irreducible facts are scattered across systems. By finding better ways for representing irreducible facts, one can potentially mitigate the problem of uncontrolled chain reactions caused by change.

### 5.1 Objects, Relationships, Processes, Events, and Patterns

In the real world, every object conveys information. The information content of physical objects is conveyed to us via one or more of our five senses. Objects are associated with one another. While some associations involve the passage of time, other associations, such as the relative locations of physical objects, are relationships that do not necessarily involve time. These relationships and associations are natural storehouses of information about real world objects. Further, these relationships are objects in their own right.

As an example of how objects may be natural repositories of information, consider a person who is employed by an organization. As an abstraction, this person exists as an object within the organizational structure of the organization. Let us call this object as Employee. While the notion of an Employee object is abstract, it is rooted in the real world by the existence of a physical employee, abstractly being an object we will call as Person. Carrying the abstraction, Person has the properties of having been born, being a living breathing mammal, and having a multitude of relationships to other objects in the real world. Once a person is employed by an organization, the object Person gains a relationship to Employee. The properties specific to Employee may contain team membership, and relationships to other entities may be lost or gained by joining or disjoining teams or being promoted. While these objects are concepts, abstracted from reality, they contain valuable information about behavior and structure. In a geographically dispersed environment, the informational content of these abstract objects is of special significance. For example, consulting the properties of Employee, e.g. skill-set and team membership, helps to assign responsibility to a specific set of modules that can best be implemented by that particular Employee in a decentralized work environment.

Processes are artifacts for expressing information about relationships that involve the passage of time, i.e., those that involve before and after effects. As such, the process is not only an association, but also an association that describes a causative temporal sequence and passage of time. This is also how the meaning of causality is born: the resources and the processes that create the product are its causes. A process always

makes a change or seeks information. Business process engineers use the term cycle time to describe the time interval from the beginning of a process to its end. A process, like the event it is derived from, can even be instantaneous or may continue on indefinitely. Processes that do not end, or have no known end, are called Sagas. Therefore, a process is a relationship, and also an event, which may be of finite, negligible, or endless duration.

Knowledge involves the recognition of patterns. Patterns involve structure, the concept of similarity, and the ability to distinguish between the components that form a pattern. Claude Shannon developed a quantitative measure for information content (Shannon, 1948). However, he did not describe the structure of information. For that, we must start with the concept and fundamental structure of *Pattern* and measurability in order to build a metamodel of knowledge. The integrated metamodel model of Pattern and measurability (from which the concept of "property" emerges) will enable us to integrate the three components that comprise business knowledge (inference, rules, and processes) into one indivisible whole. The interplay between objects and processes is driven by patterns. Patterns guide the creation of relationships between objects, such as the formation of a team or the modular assignment of duties within a team and across geographically distributed teams. Partnering Employee belonging to one team with that of another is caused by a skill or performance pattern that governs the relevant properties of Employee. As such, the ownership of an artifact under development is shared between Employee objects, which, at a coarser granularity, exist as a unified object we can refer to as a Composite Persona (CP) (Denny et al. 2008).

### 5.2 Perception and Information: Meaning, Measurability, and Format

When an object is a meaning, it is an abstract pattern of information. Its perception is a concrete expression of this meaning, and the same information may be perceived or expressed in many ways. Lacking perceptual information, several expressions or perceptions may all point to the same pattern of information, or meaning. In order to normalize knowledge, we must separate meaning from its expression. This may be done by augmenting our metamodel to represent entities of pure information that exist beyond

physical objects and relationships. This section will introduce three of these objects: Domain, Unit of Measure (UOM), and Format.

Unlike matter or energy, meaning is not located at a particular point in space and time; only its expression is (Verdu, 1998). All physical objects or energy manifested at a particular place at a point in time convey information, and the same meaning can occur in two different artifacts that have no spatial or temporal relationship with each other. They only share meaning, i.e., information content (Baggot, 1992). A single meaning may be characterized by multiple expressions. Differing understandings of concepts, terminology, and definitions are some of the problems that have characterize software developers working in a multi-site environment (P. Wongthongtham, E. Chang, T.S. Dillon, I. Sommerville 2006). Unlike a specific material object or a packet of energy that is bound to only a single location at a single point in time, identical information can exist at many different places at several different times. The need to understand the underlying natural structures that connect information to its physical expressions is inherent in the effort to normalize business rules.

Information mediation and expression within the real world is achieved by two metaobjects. One is intangible, emerging from the concept of measurability and deals with the amount of information that is inherent in the meaning being conveyed. The other is tangible; it deals with the format – or physical form – of expression. The format is easier to recognize, and many tools and techniques provide the ability to do so explicitly. It is much harder to recognize the domain of measurability, henceforth referred to simply as domain (Finkbeiner, 1966).

### 5.3 Measurability and Information Content

Through the behavior, or properties, of objects we observe, the information content of reality manifests itself to us. Although these are quite dissimilar qualities of inherently dissimilar objects, such as a person's weight and the volume of juice, both these values are drawn from a domain of information that contains some common behavior. This

common behavior – that each value can be quantitatively measured – is inherent in the information being conveyed by the measurement of these values, but not in the objects themselves..

### 5.3.1 Physical Expression of Domains

Domains convey the concepts of measurability and existence. They are a key constituent of knowledge. There are four fundamental domains that we will consider in this paper; two of them convey qualitative information and the other two convey quantitative information, as follows:

- Qualitative Domains, containing:
- Nominal Domains, which convey no information on sequencing, distances, or ratios. They convey only distinctions, distinguishing one object from another or a class from another (an object such as Person is distinct from an object such as Equine).
- Ordinal Domains, which not only convey distinctions between objects but also information on arranging its members in a sequence (a value is also an object, hence the concept of magnitude may be deemed to start here). Ordinal domains are a pattern of information derived from nominal domains by adding sequencing information, which makes it a subclass of nominal domains in the ontology of the meaning of measurability. However, ordinal domains posses no information regarding the magnitudes of gaps or ratios between objects (values).
- Quantitative Domains:
- Difference-Scaled Domains not only express all the information that qualitative domains convey, but also convey magnitudes of difference; they allow for measurement of the magnitude of point-to-point differences in a sequence. This makes difference-scaled domains to be a pattern of information derived from ordinal domains by adding quantitative information on differences between values in the domain, which makes it a subclass of ordinal domains in the ontology of the meaning of measurability. Difference-scaled domains are “dense” domains, that is, it will always be possible to locate a value between a pair of values in the in the domain, no matter how close the values that constitute the pair are to each other.

As such, we may conceive of difference scaled domains as being derived from ordinal domains by adding values to the domain until it becomes dense, and in doing so, it acquires new behaviors. However, difference-scaled domains convey no information about ratios between objects, because the domain does not contain a value in it that one can call nil or zero.

- Ratio-Scaled Domains perform three functions; they assist in the classification and arrangement of objects in a natural sequence, are able to measure the magnitude of differences in properties of objects, and take the ratios of these different properties. Ratio scaled domains always contain a natural zero. As such, ratio-scaled domains have all the information that difference-scaled domains convey, plus information on the nil value. This makes ratio-scaled domains a pattern of information derived from difference scaled domains by adding information, which makes it a subclass of difference scaled domains in the ontology of the meaning of measurability.

The hierarchy of domains provides the most fundamental kind of knowledge reuse. However, this information is still abstract. In order to give information a physical expression, it must be physically formatted and recorded on some sort of medium. A single piece of information must be recorded on at least one medium, and may be recorded in many different formats. For example, different types of equines may be coded as numbers (say, 1 for 'Horse' and 2 for 'Zebra'), or as letters (say H for 'Horse' and Z for 'Zebra'), or as pictures (say a brown equine for 'Horse' and a striped equine for 'Zebra'). This information could also be written as a hexadecimal code on floppy disk that only computers can read. This physical representation of information is its Format. A Format is an item of information, which may be attached to a meaning, but is a distinct component of information that should be distinguished from the abstract meaning it is attached to.

A symbol is sufficient to physically represent the information conveyed by nominal and ordinal domains. Of course, ordinal domains also carry sequencing information, and it would make sense to map ordinal values to a naturally sequenced set of symbols like

digits or letters. (If there is no limit to the number of values in an ordinal domain, obviously the set of 26 alphabets will not suffice, but numeric digits would, provided that we understand that quantitative differences between numbers are meaningless.)

Unlike qualitative domains, quantitative domains need both symbols and units of measure to physically express all the information they carry. This is because they are dense domains, i.e., given a pair of values, regardless of how close they are to each other, it is always possible to find a value in between them. A discrete set of symbols cannot convey all the information in a quantitative domain. However, numbers have this characteristic of being dense. Therefore, it is possible to map values in a dense domain to an arbitrary set of numbers without losing information. These numbers may then be represented by physical symbols such as decimal digits, roman numerals, or binary or octal numbers. There may be many different mappings between values and numbers. For example, age may be expressed in months, years, or days; a person's age will be the same regardless of the number used. To show that different numbers may express the same meaning, we need a Unit of Measure (UOM). The UOM is the name of the specific map used to express that meaning. Age in years, days, months, and hours are all different UOMs for the elapsed time domain.

Both the number and UOM must be physically represented by a symbol to physically format the information in a quantitative domain. Indeed, a UOM may be represented by several different symbols. The UOM "Dollars", for the money domain, may be represented by the symbol "$" or the text "USD". In general, a dense domain needs a pair of symbols to fully represent the information in it: a symbol for the UOM and a symbol for the number mapped to a value. We will call this pair the full format of the domain.

Domains, UOMs, and Formats are all objects that structure meaning. They are some of the components from which the very concept of knowledge is assembled. The Metamodel of Knowledge is a model of the meaning of knowledge built from abstract components.

Figure 2 depicts a semantic model. To understand the rules, we read along the connecting lines to form a sentence. Starting with "Quantitative Domain", for example, the sentence reads '(A) Quantitative Domain is expressed by 1 or many Unit(s) of Measure.' The lower limit (1) on the occurrence of Unit of Measure highlights the fact that each quantitative domain must possess at least one unit of measure. This is because the unit of measure is not optional. A quantitative value cannot be expressed unless a unit of measure can characterize it. The arrow that starts from, and loops back to, Unit of Measure reads 'Unit of Measure converts to none or at most 1 Unit of Measure.' Conversion rules, such as those for currency conversion or distance conversion, reside in the Metamodel of Knowledge. This relationship provides another example of a metaobject (since relationships are objects too), and demonstrates how a metaobject can facilitate the storage of the full set of conversion rules at a single place.

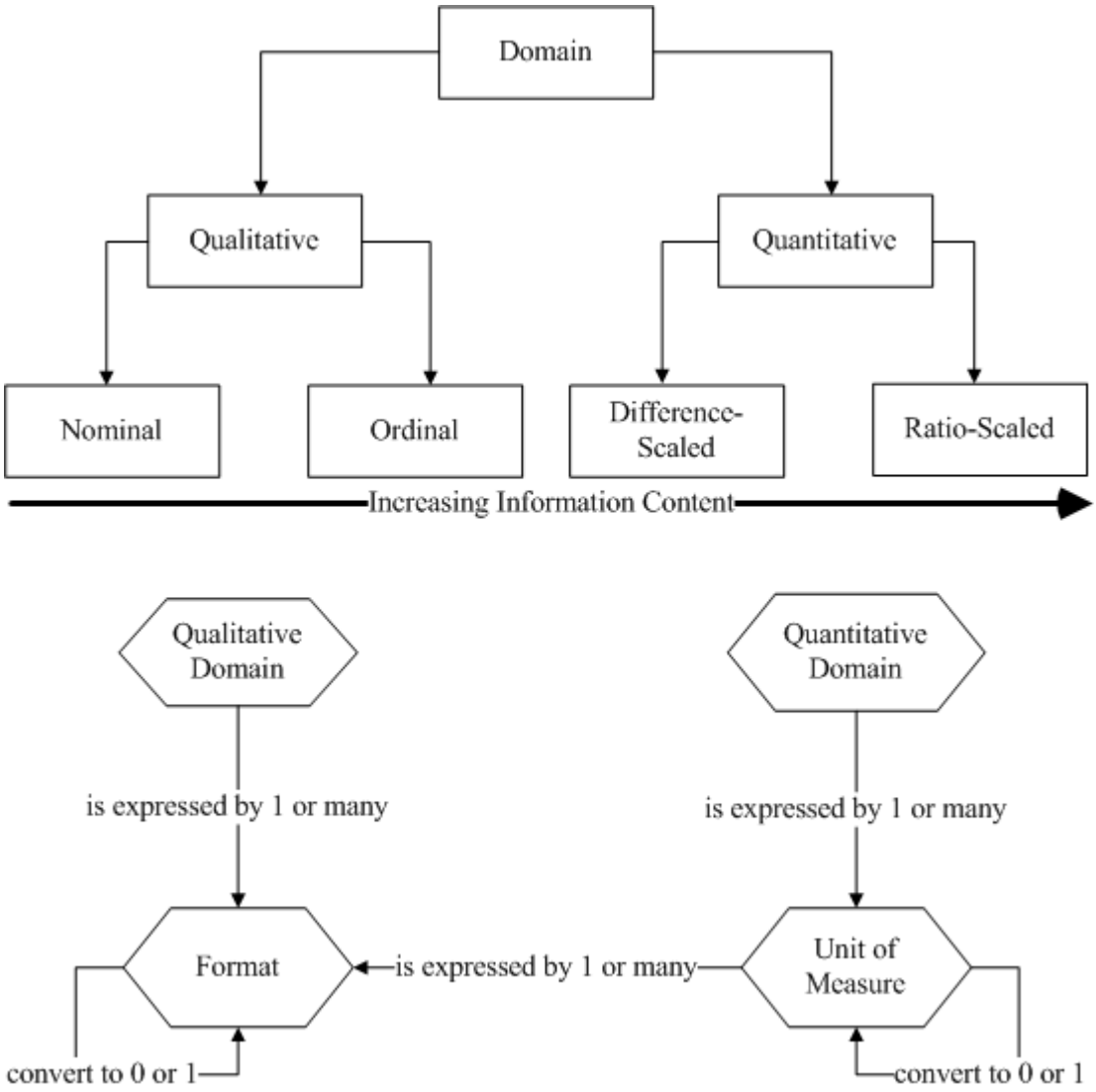



**Figure 2 A Partial Metamodel of Domain**

The conversion rule is restricted to conversion from one UOM to only one other UOM; this constraint is necessary to avoid redundancy and to normalize information. A single conversion rule enables navigation from one UOM to any other arbitrary UOM, by following a daisy chain of conversion rules. If you needed to convert yards to inches, and you had only the conversion factor to feet, you could convert yards to feet by multiplying by 3 and then to inches by multiplying by 12. The upper bound of one on the conversion relationship in the metamodel also implies that if you add a new UOM to a domain, you have to add only a single conversion rule to convert to any of the other UOMs, and that such information will suffice to enable conversion to every UOM defined for that domain.

### 5.4 Metaobjects, Subtypes, and Inheritance

Metaobjects help to normalize real world behavior by normalizing the irreducible facts we discussed earlier. The metaobjects that we have discussed so far are: object; property; relationship; process; event; domain; unit of measure (UOM); and format. The kind of atomic rules normalized by each type of metaobject are summarized in Figure 3.

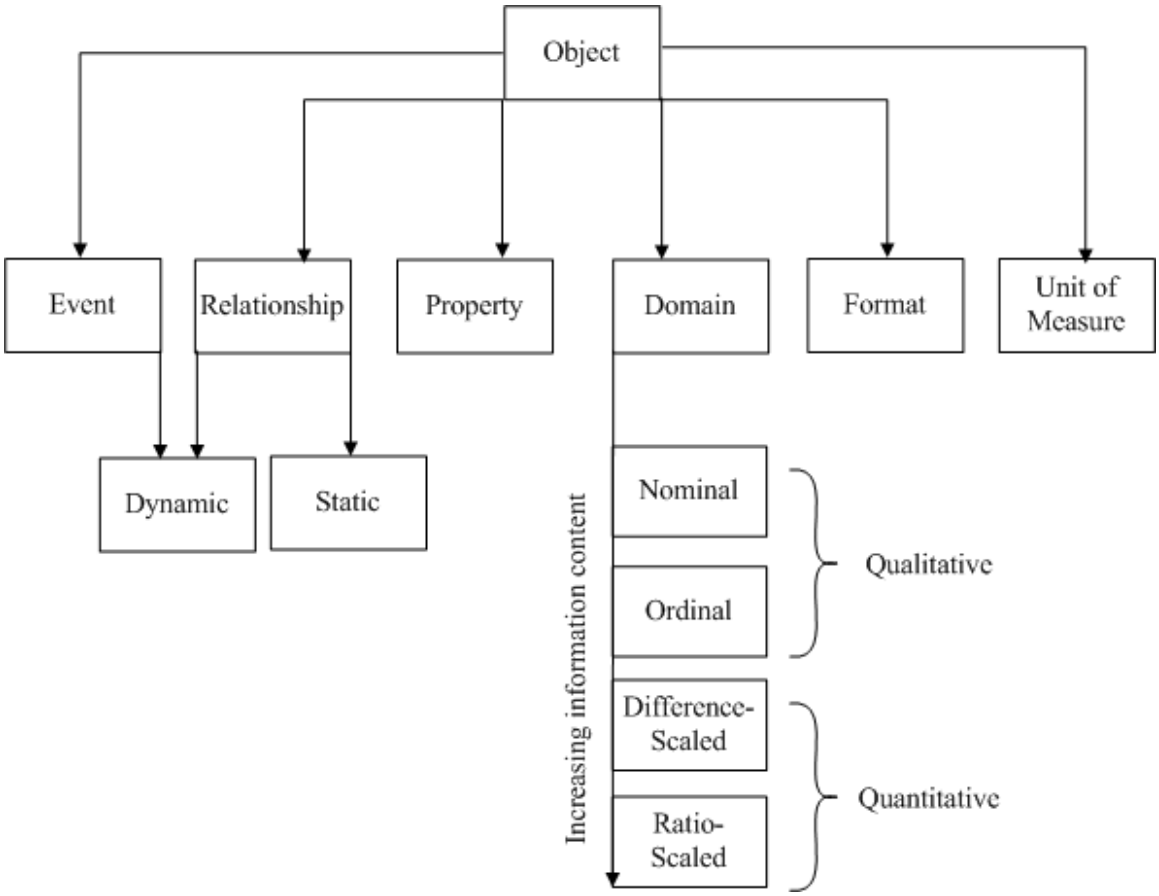



**Figure 3: Basic Inventory of Metaobjects**

The ontology in Figure 3 organizes objects in a hierarchy of meaning. Lower level objects in the ontology are derived from objects at higher levels by adding information. Figure 3 shows that the meaning of Process is configured by combining the meanings of Relationship, an interaction between objects, with the meaning of Event, the flow of time. This kind of relationship is special. It is called a subtyping relationship, and forms the basis of the ontology. Subtyping relationships convey information from higher levels to lower levels of an ontology. The lower level object becomes a special kind of higher-level object. Figure 3 shows that Ratio Scaled Domain is a special kind of Domain

because of the chain of subtyping relationships that lead from Domain to Ratio Scaled Domain via Quantitative Domain.

We now introduce two new metaobjects: the subtyping relationship and its corollary, the Subtype. They serve as containers for encapsulating and normalizing knowledge, and as conduits for sharing this knowledge with other objects. Shared behavior is normalized in the supertype object, and automatically shared with subtypes by implication, through the subtyping relationship. For example, aging, birthdays, gender, credit rating, names, ring size, social security numbers, and telephone numbers are common to all persons. People can be customers, employees, or both. The object class "Person" will normalize information common to people, such as social security number and birthday, without regard to the person being an employee, customer, or both. Subtypes will add specific information that gives the object special, more specific meanings, which are distinct and more restrictive than the meanings of their supertypes. For instance, Customer and Employee are subtypes of Person. Employee adds the employment relationship with another person or organization, while Customer has the same effect for the purchasing relationship. This is the information that Employee and Customer normalize, and add to the information conveyed by Person. They create new meanings by extending the meaning of Person. This example demonstrates why subtypes, the subtyping relationship, and inheritance are all needed to normalize information.

The information we lose when we ignore a subtyping hierarchy is information we might have reused. For example, the irreducible fact that ratio scaled values may be arranged in order of magnitude was inherited from ordinal domains. If we ignore this hierarchy in our electronic knowledge repository, we will need to replicate the comparison operators of the ordinal domain in ratio scaled domains. With the hierarchy, they will be automatically inherited.

### 5.3 The Repository of Meaning

The atomic rule is the most basic building block of knowledge, and the ultimate repository of information. It is a rule that cannot be broken into smaller, simpler parts

without losing some of its meaning. The metaobjects of Figure 3 are the natural repositories of knowledge. They provide the basis of real world meaning. Just as molecules react with molecules in chemical reactions to produce molecules of new substances with different properties from the original reagents, atomic rules may be built from other atomic rules. As we enhance our business positions with product and process innovation, some atomic rules will be reused. These rules are examples of those that can act as reusable components of knowledge. In order to build specialized domains of knowledge, entire structures and configurations may be reused. This is similar to manufacturers creating reusable subassemblies to build machines from ordinary parts. The end product may incorporate many versions and modifications of these reusable subassemblies.

### 5.4 24HrKF: A Practical Application of Knowledge Reuse

The industrial revolution led to the concepts of assembly lines, shifts for factory workers, and round-the-clock manufacturing. Advances in information systems now enable us to envision the non-stop creation of new intellectual property using shifts of workers located in different countries. More specifically, a 24-Hour Knowledge Factory (24HrKF) is envisioned as an enterprise composed of three or more sites distributed around the globe in a manner that at least one site is operational at any point of time (Gupta and Seshasai 2004). As the sun sets on one site, it rises on another with the work in progress being handed off from the closing site to the opening site.

Earlier papers on the 24-HrKF have broadly classified professional work as being ill-structured, semi-structured, or totally structured (Gupta and Seshasai 2007). CEOs, presidents, and other heads of organizations usually deal with ill-structured work, the pattern of which cannot be predicted in advance. This type of work cannot be easily decomposed into subtasks that a shadow-CEO or a partner-CEO, located in a different part of the world, can complete during the next shift. At the other end, work in industries like call centers is well-structured and can be readily picked up by a colleague coming in to work for the next shift and located in another part of the world. In between these two extremes, there are many examples of semi-structured work where the overall endeavor

can be broken into subtasks and a person in the next shift can continue to work on the incomplete task from the previous shift. Software development, in specific, and many IP based industries in general, fall into this intermediate category.

In the conventional models of software development (Beck 1999, Boehm 1988, Cockburn 2004, Coleman and Verbruggen 1998, Highsmith 2000, Poppendieck and Poppendieck 2003, Rising 2000), tasks are usually divided on a "disparate" basis among teams in different geographies; that is, one team in one part of geography developing one module, and another team in other part of the geography developing the other module. In this model, if one part of the project gets delayed, the developers working on this part end up having to devote extra hours, typically by working late into the night. Colleagues working on other parts of the project are unable to render help, because they are only familiar with their respective parts of the project. In the 24HrKF paradigm, this problem can be overcome by the "incremental" division of tasks between developers in different geographies. The latter goal can be met only with sustained research of relevant underlying issues, and the development of new agile and distributed software processes that are specially geared for use in the 24HrKF environment. New methodologies are needed to streamline the hand-off between the developer in one shift and the developer in the next shift in order to communicate details of the task in progress, as well as pending issues, in a very rapid and efficient manner in order to reduce the time spent in the hand-off to the minimum possible. Further, new task allocation processes need to be developed to address the reality that the developers will possess different skill sets and dissimilar levels of productivity.

### 5.4.1 Decomposition in the 24HrKF

The purpose of any business process is to improve the quality of the final product and increase the productivity of the participating developers. Business processes are implemented as sets of rules and procedures which guide the flow of knowledge as the project under development evolves. Using software development as an example, processes that have many rigid rules and procedures are known as 'high-ceremony.' Tayloristic processes are typically high-ceremony. Software processes that have few rules

and procedures and allow for flexibility in application of those rules and procedures are considered to be 'low-ceremony' and agile (Cockburn 2004, Highsmith 2000, Poppendieck and Poppendieck 2003, Rising 2000).

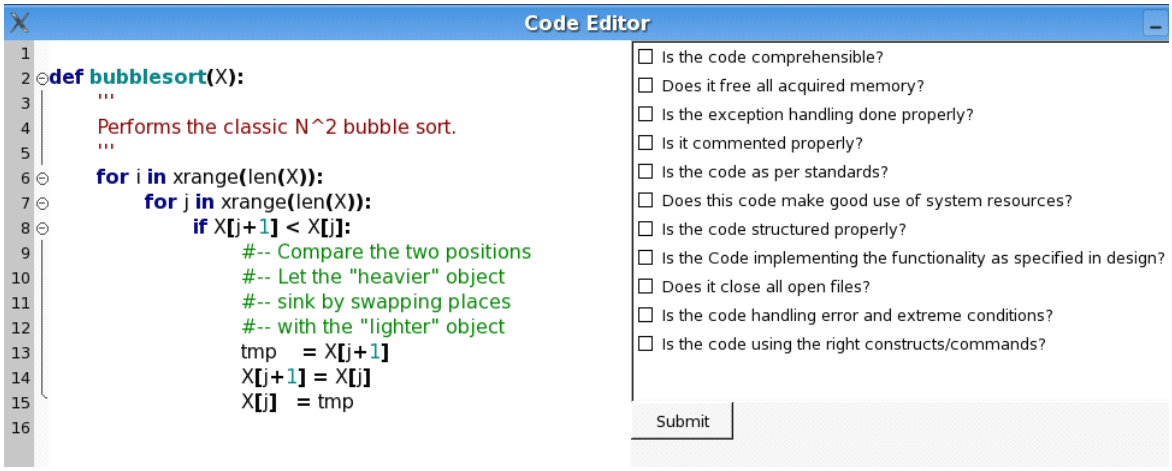
The concept that allows for lateral communication between distributed teams in the 24HrKF and facilitates knowledge reuse and agility is the Composite Persona (CP). A CP is a highly cohesive micro-team that, like a corporation, possesses simultaneous properties of both individual and collective natures. That is, a composite persona to an external observer has a unique name and acts as a singular entity, even though it is the composition of several individuals. The individual members are distributed across development sites. Using CPs, development proceeds much in the same manner as in a more traditional, local process. Problems are decomposed into modules and classes as they are when only single developers are assumed. However, when modules and classes are assigned ownership, the owner of each artifact is no longer an individual developer but rather a CP. Similarly, in the process of conflict resolution, discussion, and debate, each CP contributes as a single entity. The lateral communication between distributed teams is essentially the handoff. In a “follow the sun” setting, one team’s completion of the daily tasks inevitably leads to a period of synchronization before the next team in the handoff sequence can continue the work. In order to facilitate the handoff, the knowledge artifacts that are created or modified must be communicated to the next team in a timely and effective way. Knowledge reuse comes in the form of shared artifacts contained within a knowledge repository that records relevant developer actions.

### 5.4.2 Knowledge Reuse in the 24HrKF

Suchan and Hayzak (2001) found that a semantically rich database was useful in creating a shared language and mental models. Multimind is a novel collaboration tool under development and experimentation at the University of Arizona; it aims to provide a semantically rich environment for developers collaborating using the CP method (Denny et al. 2008). MultiMind aims to improve upon DiCE (Vin at al, 1993) and other collaborative engineering tools.

Objects and events relevant to the project are posted and logged into a chronologically ordered persistent database. This database, sometimes called a Lifestream (Freeman and Gelernter 1996), incorporates an algebra that can be coupled to the semantics of project artifacts and knowledge events to enable substantial automation of both mundane tasks and higher-level functions. In the MultiMind tool, the Lifestream is used to archive project artifacts and provide configuration management services in much the same fashion as the Concurrent Versioning System (CVS) or the newer Subversion (SVN) revision control system. MultiMind also observes and logs knowledge events into the LifeStream. When a developer reads a message, posts a message, executes a search or reads a web page, MultiMind logs this activity into the LifeStream. MultiMind can therefore correlate knowledge events with the discrete evolutions of the project artifacts, which allows for the reuse of relevant project knowledge between members of a CP.

Actions recorded by Multimind facilitate communication by encapsulating knowledge as objects representing interactions with the developer's environment. Developers in all distributed locations can query Multimind to find the current state of each artifact under their CP's. More importantly, developers can also manually update the state of each artifact. An example is shown in the following figure.




In this case, a code review checklist is presented along with the relevant code. A developer performing the code review would generate responses to those queries that cannot be automated. In this example, additional relevant review entries can be added to

further improve communication of salient knowledge items between teams. In the handoff, the use of lightweight Scrum methods is prevalent. A Scrum is a stand-up meeting pattern that attempts to ascertain the progress that was made in the prior shift, the problems encountered in the prior shift, and the progress that will be made in the current shift.

Automated and manual information gathering mechanisms can further be combined to provide knowledge artifacts beyond communication facilitating ones. For example, visualizing the current project state is simplified through use of Multimind objects. Since it relies on LifeStream, which contains chronological information regarding each artifact, a visualization of a project's progress can easily be made, identifying those artifacts that required the most maintenance, debugging, or synchronization from the CP. Business decisions can be facilitated with use of Multimind's LifeStream visualizations. Filtering the Multimind queries to those artifacts that are relevant to a particular decision makes this facilitation possible. Communication, visualization, and decision facilitation are examples of knowledge reuse in the 24HrKF.

## 6 An Architecture of Knowledge

Information Systems are built to satisfy business requirements. Sometimes they are undertaken to implement purely technical changes (Smith and Fingar 2002). Although requirements are the critical components of any information system, to this day, almost half a century after business process automation first appeared, "Requirement" has remained a nebulous concept in the industry. There is no common understanding, let alone agreement, on what requirements really mean. Poorly formulated and ill-managed requirements have led to many of the problems that Information Systems projects currently face (Bahill, A.T. and Dean, F. (1999). Our first task, therefore, is to understand the meaning and structure of requirements.

Requirements flow from knowledge. Knowledge is encapsulated in configurations of atomic rules. Knowledge of Information Systems involves configurations of (atomic)

rules of business as well as technology. A four-layered hierarchical approach can be used, as depicted in Figure 4.

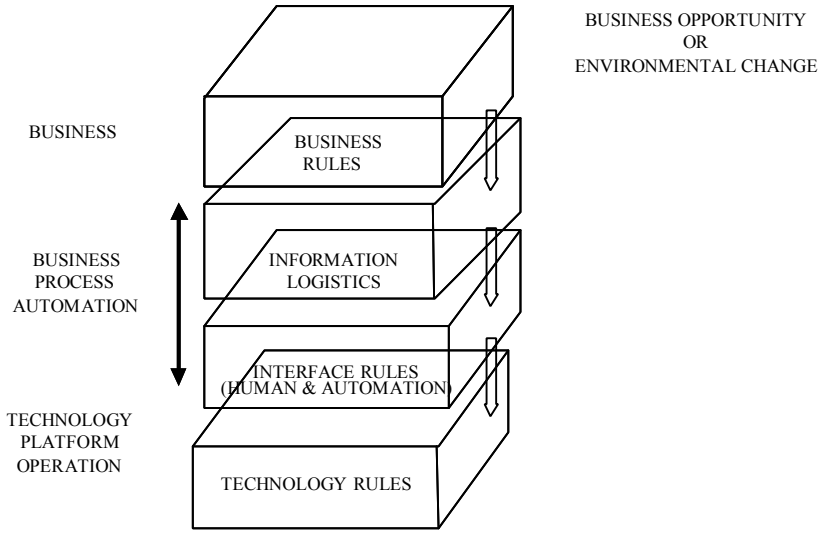



Figure 4: The Architecture of Knowledge

Table 1 contains brief descriptions of each layer of the architecture of business knowledge, examples of the kinds of policies that may be implemented at the layer, as well as examples of change that may occur in that layer along with examples of the effects of change within a particular layer.

| **Layer** | **Description** | **Example Policy** | **Example of Change** |
|---|---|---|---|
| Business | Contains assertions about products and services and defines relationships between customers and products or services. | Obtain necessary market freedoms to effectively compete; Purchase competitor fixed assets; Penetrate untapped markets | Acquisition of new assets necessitates the addition of new relationships between customers and services. |
| Information Logistics | Contains the repository of rules related to the logistics of storage, transfer, and utilization of business infor | Digital artifacts under development will be stored using a versioning system; | A new employee joins an active team necessitating a change in the rules regarding a |

| | | | |
|---|---|---|---|
| | mation. | Artifact access privileges are maintained at a fine granularity | ccess to a team-owned artifact. |
| Interface | Contains the rules for the presentation of business information to human entities. | Access to team-owned objects is controlled by an administrator in close contact with team members; GUI follows Microsoft's Inductive User Interface guidelines. | A new employee joins an active team, necessitating the creation of additional security rules regarding artifact access privileges. |
| Technology | Contains low-level operational and strategic rules regarding the operation of technology. | Hardware and systems software is standardized through a single reputed vendor. | A change of hardware or systems software vendor necessitates change of legacy software developed for obsoleted system. |

The top business layer helps to assemble components of knowledge into business concepts, such as products, services, markets, regulations, and business practices. Consider a situation where a telephone services provider wishes to integrate cable TV and entertainment media into its business. Such changes in the Business Rules layer will impact business functions and systems functionality, whereas changes to process automation layers alone will impact only availability, timeliness, accuracy, reliability, and presentation of information. Changes in business process automation, in turn, can impose new requirements for performance, reliability and accuracy on technology platforms, which will impact the technology layer.

The level of Business Process Automation is usually changed to leverage information technology or to focus on those processes that create most value while eliminating those of little value. Changes in this layer seldom impact the fundamental business of the firm. For example, the firm could deploy its ordering process on the web, but not make any fundamental change in the nature of its products, services, or markets. Business Process Automation refers to process innovation and change that leverages information technology.

The technology layer is changed primarily to improve computer performance in terms of speed, cost, reliability, availability or alignment, and support for Business Process Automation.

The fundamental ideas of separating system-specific rules from software implementation, as in the case of 3GL, and separating system architecture and implementation, as in the case of System/360, are even more important today in the context of separating business rules from implementation technologies. The rules related to transporting and presenting the information would belong to the Business Process Automation layers, not the pure business layer. Figure 4 shows that Business Process Automation consists of two layers. The Information Logistics layer is the repository for rules related to the logistics of moving and storing information in files, and the Interface layer is concerned with how this information is presented to human operators.

Creating a business knowledge hierarchy such as the one depicted in Figure 4 facilitates the flow of information between the business entities responsible for managing knowledge. Organizing knowledge and information, as described in previous sections, is essential for realizing the flow of information between the layers and creating meaningful and useful relationships between objects within each layer.

Efforts to process and integrate information based on meaning have been made by the W3C consortium, which recommended two modeling standards in 2004: RDF, the

Resource Description Framework for metadata, and OWL, the Web Ontology Language for integrating information. We've seen examples of how some meanings are derived from others by constraining patterns of information they convey to create new meanings. These constrained patterns are subtypes of the meanings they constrain, and every meaning is a polymorphism of the universal object, an unknown pattern in information space that means everything and anything, and conveys nothing. Every object in the inventory of components is a polymorphism of the universal metaobject. RDF and OWL are tailored for the Web and applications of the Semantic Web. Tables 2, 3, and 4 show that the various elements of RDF and OWL as well as their Metaobject Inventory equivalents, showing that, in effect, the Metaobject Inventory provides a more general framework than either RDF or OWL, and that either of the restricted frameworks are special cases of the types of ontology frameworks that can be realized through the various polymorphisms of the universal object.

| **Element** | **Class of** | **Subclass of** | **Metaobject Inventory Equivalent** |
|---|---|---|---|
| Class | All classes | | All value |
| Datatype | All Data types | Class | Domain, Meaning |
| Resource | All resources | Class | Resource |
| Container (set of objects) | All Containers | Resource | Aggregate Object |
| Collection(set membership is restricted by some criteria) | All Collections | Resource | Object Class |
| Literal | Values of text and numbers | Resource | Subtype of Symbol |
| List | All Lists | Resource | List of |
| Property | All Properties | Resource | Property, Feature |
| Statement | All RDF Statements | Resource | Irreducible fact, rule, atomic rule |
| Alt | Containers of alternatives | Container | Mutability; Liskov's principle, aggregation of mutable resources |
| Bag | Unordered containers | Container | Aggregate Object |
| Seq | Ordered containers | Container | Subtype of Aggregate Object |
| ContainerMembershipProperty | All Container membership properties | Property | Subtype of Relationship |
| XMLLiteral | XML literal values | Literal | Subtype of symbol. XML is a subtype of language. |

Table 2 RDF Classes and their metaobject inventory equivalents.

| Property | Operates on | Produces | Description | Metaobject Inventory Equivalent |
|---|---|---|---|---|
| Domain | Property | Class | The domain of the resource The domain defines what a property may apply to (operate on). | Domain |
| Range | Property | Class | The range of the resource. It defines what the property may map to (produce). | co-domain |
| subPropertyOf | Property | Property | The property of a property | Feature |
| subClassOf | Class | Class | Subtyping property | Polymorphism |
| Comment | Resource | Literal | User friendly resource description | Elaboration, description, syn |
| Label | Resource | Literal | User friendly resource name | Name, synonym |
| isDefinedBy | Resource | Resource | Resource definition | Id |
| seeAlso | Resource | Resource | Additional information about a resource | Elaboration, reference |
| Member | Resource | Resource | The property of being an instance of a kind of resource | Instance of |
| First | List | Resource | The property of being the first member of a list | A demiliting role: Lower Lir |
| Rest | List | List | The second and subsequent members of a list | Subtype of List |
| Subject | Statement | Resource | The subject of an assertion, i.e., the subject of a resource in an RDF statement | The source of a relationship |
| predicate | Statement | Resource | Similar to "subject": The predicate of an assertion | Relationship, function |
| object | Statement | Resource | The object of the resource (in an RDF) Statement | The target of a relationship |
| value | Resource | Resource | The value of a property | Value |
| Type | Resource | Class | An instance of a class | Member of a class of classes |

Table 3 RDF Properties and their Metaobject Inventory equivalents

| Class | Description | Metaobject Inventory Equivalent |
|---|---|---|
| AllDifferent | all listed individuals are mutually different | Subtype of Exclusion partition, exclusivity constraint. The concept of distinctions emerges as a polymorphism of the concept of class as information is added o an object/pattern. |
| allValuesFrom | All values of a property of class X are drawn from class Y (or Y is a description of X) | Domain, Inclusion Set, inclusion partition |
| AnnotationProperty | Describes an annotation. OWL has predefined the following kinds of annotations, and users may add more:<br>• Versioninfo<br>• Label<br>• Comment<br>• Seealso<br>• Isdefinedby<br>OWL DL limits the object of an annotation to data literals, a URIs, or individuals (not an exhaustive set of restrictions | Subtypes of Elaboration<br><br>Version is implicit in temporal objects. Audit properties are implicit in object histories:<br>• The process, person, event, rule, reason and automation that caused a state to change<br>• Time of state change<br>• Who made the change (All the dimensions of process ownership: Responsibility, Authority, Consultation, Work, Facilitation, Information/knowledge of transition)<br>• When the change was made<br>• The instance of the process that caused the change and the (instances of resources) that were used<br>• Why it was made (the causal chain that led to the process)<br>• How long it took to make the change<br><br>Label is implicit in synonym, name<br>Comment may be elaboration or reference. The two are distinct in the metamodel of knowledge<br>See also: same remarks as comment.<br>IsDefinedBy may be elaboration, Object ID, or existence dependency. Each is a distinct concept in the metamodel of knowledge |

| Class | Description | Metaobject Inventory Equivalent |
|---|---|---|
| backwardCompatibleWith | The ontology is a prior version of a containing ontology, and is backward compatible with it. All identifiers from the previous version have the same interpretations in the new version. | Part of Relationship between models or structures |
| cardinality | Describes a class has exactly N semantically distinct values of a property (N is the value of the cardinality constraint). | Cardinality |
| Class | Asserts the existence of a class | Object Class |
| complementOf | Analogous to the Boolean "not" operator. Asserts the existence of a class that consists of individuals that are NOT members of the class it is operating on. | Set negation, Excludes, Exclusion set, Exclusion partition |
| DataRange | Describes a data type by exhaustively enumerating its instances (this construct is not found in RDF or OWL Lite) | Inclusion set, exhaustive partition |
| DatatypeProperty | Asserts the existence of a property | Feature, relationship with a domain |
| DeprecatedClass | Indicates that the class has been preserved to ensure backward compatibility and may be phased out in the future. It should not be used in new documents, but has been preserved to make it easier for old data and applications to migrate to the new version | Interpretation. However, the specific OWL interpretation of depreciated class is considered to be a physical implementation of a real life business meaning, outside the scope of a model of knowledge that applies on the plane of pure meanings. |
| DeprecatedProperty | Similar to depreciated class | See Depreciated Class |
| differentFrom | Asserts that two individuals are not the same | The concept of distinctions emerging as a polymorphism of the concept of class as information is added o an object/pattern.; subtype of exclusion partition |
| disjointWith | Asserts that the disjoint classes have no common members | Exclusion partition |
| distinctMembers | Members are all different from each other | Exclusion Set, List |
| equivalentClass | The classes have exactly the same set of members. This is subtly different from class equality, which asserts that two or more classes have the same meaning (asserted by the "sameAs" construct). Class equivalence is a constraint that forces members of one class to also belong to another and vice-versa. | Mutual inclusion constraint/equality between partitions or objects. |
| equivalentProperty | Similar to equivalent class: i.e., different properties must have the same values, even if their meanings are different (for instance, the length of a square must equal its width). | Equality constraint |
| FunctionalProperty | a property that can have only one, unique value. For example, a property that restricts the height to be non-zero is not a functional property because it maps to an infinite number of values for height. | Value of a property, singleton relationship between an object and the domain of a property |
| hasValue | Links a class to a value, which could be an individual fact or identity, or a data value (see RDF data types) | relationship with a domain |

| Class | Description | Metaobject Inventory Equivalent |
|---|---|---|
| imports | References another OWL ontology. Meanings in the imported ontology become a part of the importing ontology. Each importing reference has a URI that locates the imported ontology. If ontologies import each other, they become identical, and imports are transitive. | Subtype of Composed of. Note that the metamodel of knowledge does not reference URIs. This is an implementation specific to the Web. The Metamodel of Knowledge deals with meanings. |
| incompatibleWith | The opposite of backward compatibility. Documents must be changed to comply with the new ontology. | Reinterpretation, Intransitive Relationship, asymmetrical relationships |
| intersectionOf | Similar to set intersection. Members are common to all intersecting classes. | Subtype of Partition, subtype with multiple parents, set intersection |
| InverseFunctionalProperty | Inverses must map back to a unique value. Inverse Functional properties cannot be many-to-one or many-to-many mappings | Inverse of an injective or bijective relationship |
| inverseOf | The inverse relationship (mapping) of a property from the target (result) to the source (argument) | Inverse of |
| maxCardinality | An upper bound on cardinality (may be "many", i.e., any finite value) | Cardinality constraint:, upper bound on cardinality (subtype of cardinality constraint and upper bound) |
| minCardinality | A lower bound on cardinality | Cardinality constraint: Lower bound on cardinality (subtype of cardinality constraint and lower bound) |
| Nothing | The empty set | of the empty set, null value |
| ObjectProperty | Instances of properties are not single elements, but may be subject-object pairs of property statements, and properties may be subtyped (extended). ObjectProperty asserts the existence and characteristics of properties:<br>• RDF Schema constructs: rdfs:subPropertyOf, rdfs:domain and rdfs:range<br>• relations to other properties: owl:equivalentProperty and owl:inverseOf<br>• global cardinality constraints: owl:FunctionalProperty and owl:InverseFunctionalProperty<br>• logical property characteristics: owl:SymmetricProperty and owl:TransitiveProperty | Property, a generalized constraint, which implies an information payload added to a meaning. |
| oneOf | The only individuals, no more and no less, that are the instances of the class | members of a class, the property of exhaustivity of a partition |
| onProperty | Asserts a restriction on a property | constraint on a Feature (makes the feature (object) a subtype of the unconstrained, or less constrained feature (object) |
| Ontology | An ontology is a resource, so it may be described using OWL and non-OWL ontologies | The concept of deriving subclasses by adding information to parent classes |
| OntologyProperty | A property of the ontolology in question. See imports. | None, beyond the fact that the ontologoly is an object, which means that it inherits all properties of objects, and adds the property of interpretation |

| Class | Description | Metaobject Inventory Equivalent |
|---|---|---|
| priorVersion | Refers to a prior version of an ontology | An instance of Object Property where a relevant instance of ontology Object Class exists, containing a Temporal Succession of concepts. The property of reinterpretation is implicit between versions of an ontology. |
| Restriction | Restricts or constrains a property. May lead to property equivalence, polymorphisms, value constraints, set operations, etc. | Rule Constraint |
| sameAs | Asserts that individuals have the same identity. Naming differences are merely synonyms | Set Equality, Identity |
| someValuesFrom | Asserts that there exists at least one item that satisfies a criterion. Mathematically, it asserts that at least one individual in the domain of the "SomeValuesFrom" operator that maps to the range of that operator. | Subsetting constraint |
| SymmetricProperty | When a property and its inverse mean the same thing (e.g., if Jane is a relative of John, then John is also a relative of Jane) | Symmetry |
| Thing | The set of all individuals. | Instance of Object Class |
| TransitiveProperty | If A is related to B via property P1, and B is related to C via property P2, then A is also related to C via property P1. For example. If a person lives in a house, and the house is located in a town, it may be inferred that the person lives in the town because "Lives in" is transitive with "Located in". | Transitive Relationship |
| unionOf | Set union. A member may belong to any of the sets in the union to be a member of the resulting set | offset Untion, Aggregation |
| versionInfo | Provides information about the version | Instance of Attribute. Implicit in the concept of the history of a temporal object |

Table 4 OWL Classes and their Metaobject Inventory equivalents.

## 7 Conclusions

In an effort to provide a framework for surmounting the temporal and spatial separations in collaborative, distributed environments, this paper presents a framework for knowledge object management that facilitates knowledge reuse. Specific cases of practical applications of knowledge reuse are given through discussions of early industry efforts to this end and the 24-Hour Knowledge Factory concept currently under development. The encapsulation of knowledge for distributed environments is also shown to be essential in the proposed architecture of business knowledge which in turn facilitates the understanding of concepts and terminologies used by different teams. The

knowledge encapsulation allows for a four-tier architecture that facilitates knowledge transfer and reuse while facilitating the understanding of the problem domain under consideration. Differences in training, knowledge and skills that exist across the distributed teams are surmounted by use of a common means of discourse about the problem domain under consideration provided by the tools and mechanisms developed for the 24-Hour Knowledge Factory.

## 8 For Further Reading

The concepts described here have been utilized and extended in this paper to cater specifically to the special needs of offshoring and 24-Hour Knowledge Factory environments. For a detailed discussion of the basic concepts and their wider applications, please refer to the following books by Amit Mitra and Dr. Amar Gupta:

- Agile Systems with Reusable Patterns of Business Knowledge - a Component Based Approach
- Creating Agile Business Systems with Reusable Knowledge
- Knowledge Reuse and Agile Processes – Catalysts for Innovation

## 9 References


Aggarwal, A. and Pandey, A. (2004) Offshoring of IT Services – Present and Future. Evalueserve, New Delhi. (http://www.evalueserve.com)

Amdahl, G.M., Blaauw, G.A., and Brooks, F.P. Jr. (2000) Architecture of the IBM System/360, (44:1/2), pp 21.

Baldwin, C.Y. and Clark, K.B. (2000) *Design Rules, Vol. 1: The Power of Modularity*, The MIT Press, Cambrige, Mass.

Baggot, Jim. (1992). The Meaning of Quantum Theory, Oxford University Press.

Bahill, A.T. and Dean, F. (1999). Discovering system requirements, Handbook of Systems Engineering and Management, Chapter 4, John Wiley & Sons, 175-220.

Beck, K. (1999) *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Mass..

Boehm, B. (1988) A Spiral Model of Software Development and Enhancement. Computer (21:5), pp. 61-72.

Blanchard, Eugene. (2001). Introduction to Networking and Data Communications. Commandprompt, Inc.

Business Process Notes, retrieved from: http://www.w3.org/2004/12/rules-ws/paper/105/

Carmel, E. (1999) *Global Software Teams: Collaborating Across Borders and Time Zones*, Prentice Hall, 1999.

Chang, E., Dillon T.S., Sommerville, I, Wongthongtham, P. (2006). Ontology-based multi-site software development methodology and tools. Journal of Systems Architecture, Volume 52, Issue 11.

Cockburn, A. (2004) *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional.

Coleman, G. and Verbrugge, R. (1998) A Quality Software Process for Rapid Application Development. Software Quality Journal (7). pp. 107-122.

Danait, A. (2005). "Agile Offshore Techniques – A Case Study", in Proceedings of the IEEE Agile Conference **,** pp. 214-217.

Denny, N., Mani, S., Sheshu, R., Swaminathan, M., Samdal, J. and Gupta, A. (2008). Hybrid Offshoring: Composite Personae and Evolving Collaboration Technologies. To appear in *Information Resources Management Journal*, 2008.

Finkbeiner, Daniel. (1966). Matrices and Linear Transformations, Chapter 1. W. H. Freeman and Co..

Freeman, E. and Gelertner, D., "Lifestreams: A Storage Model for Personal Data," ACM SIGMOD Record (25:1), March 1996, pp. 80-86.

Gupta, A., Seshasai, A., Mukherji, S., Ganguly, A. (2007 April-June). Offshoring: The Transition from Economic Drivers Toward Strategic Global Patnership and 24-Hour Knowledge Factory. Journal of Electronic Commerce in Organizations, 5(2), 1-23.

Gupta, A. and Seshasai, S. (2007) "24-Hour Knowledge Factory: Using Internet Technology to Leverage Spatial and Temporal Separations," ACM Transactions on Internet Technology, Vol. 7, No. 3, August 2007.

Highsmith, J.A. (2000) *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House, New York.

Kanka, Manfred. (2001). A Paper on Semantics. Institut für deutsche Sprache und Linguistik, HU Berlin.

Kingston, J. (2002) Merging Top Level Ontologies for Scientific Knowledge Management. *Proceedings of the AAAI Workshop on Ontologies and the Semantic Web*. Retreived from http://www.inf.ed.ac.uk/publications/report /0171.html

Kussmaul, C., Jack, R., Sponsler, B. (2004). "Outsourcing and Offshoring with Agility: A Case Study", in Zannier, C. et al. (eds.) *XP/Agile Universe 2004, LNCS 3132*, 147-154, Springer.

López-Bassols, Vladimir. (1998). Y2K. The OECD Observer, No. 214.

Markus, L.M. (2001) Toward a Theory of Knowledge Reuse: Types of Knowledge Reuse Situations and Factors in Reuse Success, Journal of Management Information Systems (18:1), pp. 57-93.

Mitra, A., & Gupta, A. (2005). Agile Systems with Reusable Patterns of Business Knowledge. Artech House

Mitra, A., & Gupta, A. (2006). Creating Agile Business Systems with Reusable Knowledge. Cambridge, University Press.

Myopolous, J. (1998). Information Modeling in the Time of Revolution. *Information Systems* 23 (3-4).

Ravichandran, T. (2005). Organizational Assimilation of Complex Technologies: An Empirical Study of Component-Based Software Development. IEEE Transactions on Engineering Management, vol. 52, no. 2

Fergusson, N., "Survival of the Biggest," Forbes 2000, April 12, 2004, p. 140.

O'Leary, D.E. (2001) How Knowledge Reuse Informs Effective System Design and Implementation, IEEE Intelligent Systems (16:1), pp 44-49.

Poppendieck, M. and Poppendieck, T. (2003) *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison-Wesley Professional.

Rising, L., and Janoff, N. S. (2000) The Scrum software development process for small teams, Software, IEEE (17:4), pp. 26-32.

Ross, R. G. (1997). The Business Rule Article: Classifying, Defining and Modeling Rules. Database Research Group.

Sparling, M. (2000) Lessons learned through six years of component-based development. Communnications of the ACM, vol. 43, no. 10, pp. 47–53.

Suchan and Hayzak (2001) The communication characteristics of virtual teams: a case study. IEEE Transactions on Professional Communication. v44 i3. 174-186.

Smith, H. and Fingar, P. (2002) The Next Fifty Years, Darwin, retrieved from http://www.darwinmag.com/read/120102/bizproc.html

Szyperski, C. (1997) Component Software: Beyond Object-Oriented Programming. ACM Press.

Treinen J. J. and Miller-Frost, S. L. (2006) "Following the sun: Case studies in global software development," IBM Systems Journal, Vol. 45, No. 4, 2006.

Vanwelkenhuysen, J. and R. Mizoguchi (1995) Workplace-Adapted Behaviors: Lessons Learned for Knowledge Reuse, Proceedings of KB&KS, pp 270-280.

Van Zyl, J. and Corbett, D. (2000) Framework for Comparing Methods for using or Reusing Multiple Ontologies in an Application. *Proceedings of the 8th International Conference on Conceptual Structures*.

Verdu, Sergio. (1998). IEEE Transactions on Information Theory, 44 (6).

Vin, H.M., Chen, M.-S., Barzilai, T. (1993) Collaboration Management in DiCE, The Computer Journal, 36(1):87-96.

Yap, M. (2005) "Follow the Sun: Distributed Extreme Programming Environment," Proceedings of the IEEE Agile Conference, pp 218-224.

Xiahou, Y., Bin, X., Zhijun, H. and Maddineni, S. (2004) "Extreme Programming in Global Software Development," Canadian Conference on Electrical and Computer Engineering, Vol. 4, May 2004.